PROGRAMA EDUCATIVO
INFORMATICA ADMINISTRATIVA

UNIDAD DE APRENDIZAJE
BASES DE DATOS RELACIONALES

Unidad de competencia IV

Aprender el funcionamiento de la implementación de una base de datos relacional conociendo álgebra y cálculo relacional para llevar acabo la administración y manejo de la información mediante un sistema manejador de base de datos relacional (SQL)

Aprender a aplicar el álgebra y calculo relacional con SQL

ELABORACION
ADRIAN TRUEBA ESPINOSA
Centro Universitario UAEM Texcoco
Informática Administrativa
Promoción 2016B

PRESENTACIÓN DEL CURSO

La unidad de aprendizaje “Bases de Datos Relacionales”, se imparte en el 7° semestre de la licenciatura en Informática administrativa. Tiene la finalidad de desarrollar las competencias necesarias en los alumnos, para la implementación de bases de datos a través del modelo de bases de datos relacional. Para ello es necesario sentar las bases teóricas y metodológicas para el diseño de una base de datos relacional.

CONTENIDO DEL CURSO

Unidad I. Identificar los conceptos principales de bases de datos relacionales

Unidad II. - Aprender los principios fundamentales para el diseño de una base de datos bajo el modelo relacional para poder aprovechar sus beneficios respecto a la integridad y consistencia de datos

Unida III. -Aprender el proceso de normalización en el modelo relacional para hacer eficaz y confiable la información del ente económico mediante el modelo relacional

Unidad IV. Aprender el funcionamiento de la implementación de una base de datos relacional conociendo álgebra y cálculo relacional para llevar acabo la administración y manejo de la información mediante un sistema manejador de base de datos relacional (SQL)

Unidad V. Identificar los diferentes lenguajes de consulta de gráficos para realizar manipulación de información desde dichos lenguajes

Unidad VI. Aprender los principios de sistemas de bases de datos distribuidas para implementar bases de datos relacionales y cumplir con estrategias y objetivos de la empresa

METAS A ALCANZAR

Que el alumno desarrolle las competencias técnicas y profesionales para el desarrollo e implementación de bases de datos tales como:

•Conocer los elemento teóricos de lenguaje de consultas a bases de datos Algebra y calculo relacional.

•Conocer los elementos teóricos del lenguaje SQL

OBJETIVO DEL MATERIAL DIDÁCTICO

Que el alumno conozca los lenguajes de consulta a bases de datos relacionales con álgebra y calculo relacional

METODOLOGÍA DEL CURSO

El curso se desarrollará bajo el siguiente proceso de estudio:

1. Exposición de parte del profesor mediante la utilización de este material en diapositivas.
2. Control de lecturas selectas que el profesor asignará para complementar la clase.
3. Tareas donde se investigarán temas, conceptos, procesos y métodos de los temas por ver.
4. Participación en clases
5. Prácticas de laboratorio

UTILIZACIÓN DEL MATERIAL DE DIAPOSITIVAS

El material didáctico visual es una herramienta de estudio que sirve como una guía para que el alumno repase los temas más significativos del “Método de implementación de base de datos relacional Lenguajes de consulta gráficos ”, cabe aclarar que será un tutor el cual proporcionará las ideas generales del tema, asiendo ejercicios en el salón de clase.

UNIDAD DE APRENDIZAJE
BASES DE DATOS RELACIONALES

Unidad de competencia IV

Aprender a aplicar el álgebra y calculo relacional con SQL

Lenguaje de datos

El lenguaje de definición de datos, denominado por sus siglas como: DDL(Data definition Language).

Permite definir un esquema de base de datos por medio de una serie de definiciones que se expresan en un lenguaje especial, el resultado de estas definiciones se almacena en un archivo especial llamado diccionario de datos.

Lenguaje de datos:

Es un lenguaje de manejo de datos para el sistema relacional, el álgebra relacional y cálculo relacional, ambos lenguajes son "relacionalmente completos", esto es, cualquier relación que pueda derivarse de una o más tablas de datos, también se puede derivar con u solo comando del sublenguaje. Por tanto, el modo de operación de entrada/Salida en un sistema relacional se puede procesar en la forma: una tabla a la vez en lugar de: un registro a la vez; en otras palabras, se puede recuperar una tabla en vez de un solo registro con la ejecución de un comando del sublenguaje de datos.

Lenguaje de manipulación de datos

La manipulación de datos se refiere a las operaciones de insertar, recuperar, eliminar o modificar datos; dichas operaciones son realizadas a través del lenguaje de manipulación de datos (DML, Data Manipulation Language), que es quién permite el acceso de los usuarios a los datos. Existen básicamente 2 tipos de lenguajes de manipulación de datos:

Lenguajes de consulta formales.

Los lenguajes de consultas:

Son los lenguajes en el que los usuarios solicitan información de la base de datos. Estos lenguajes son generalmente de más alto nivel que los lenguajes de programación. Los lenguajes de consulta pueden clasificarse como ***procedimentales*** y ***no procedimentales***;

El álgebra relacional es un lenguaje de consulta formal procedimental, el álgebra relacional define operadores que funcionan sobre las tablas (de una manera similar a los operadores +,-,etc. del álgebra común) para llegar al resultado deseado. El álgebra relacional es difícil de utilizar, debido en parte a que es procedimental, esto es, al utilizar el álgebra relacional no sólo debemos ***saber*** lo que queremos, también ***cómo*** obtenerlo.

- **Procedimentales:**

  Los LMD requieren que el usuario especifique que datos se necesitan y cómo obtenerlos

- **No procedimentales:**

  Los LMD requieren que el usuario especifique que datos se necesitan y sin especificar cómo obtenerlos.

En el lenguaje del tipo *procedimental* el usuario da las instrucciones al sistema para que realice una secuencia de operaciones en la base de datos para calcular el resultado deseado.

En el lenguaje *no procedimental,* el usuario describe la información deseada sin dar un procedimiento específico para obtener dicha información.

En el proceso de bases de datos comerciales el álgebra relacional se utiliza de manera poco frecuente. Aunque unos cuantos productos exitosos DBMS sí tienen opciones del álgebra relacional, éstas son poco utilizadas en vista de su complejidad.

El álgebra relacional toma dos o más tablas como entrada produce una nueva tabla como resultado de la serie de operaciones.

Las operaciones fundamentales en el álgebra relacional son *seleccionar, proyectar, producto cartesiano, renombrar, unión y diferencia de conjuntos.*

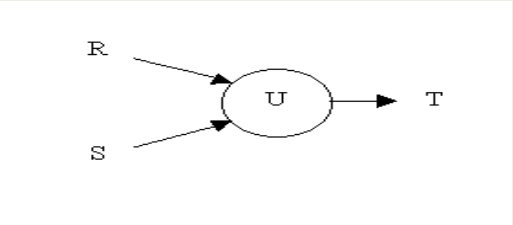
Además de las operaciones fundamentales existen otras operaciones como son: *intersección de conjuntos, producto natural, división y asignación*

Lenguajes de manipulación de datos para BDR

Álgebra relacional

Es una colección de operaciones formales sobre las relaciones. Las operaciones básicas son de dos tipos: unarias y binarias.

Unión: (∪) La unión de dos relaciones R y S con el mismo esquema es una relación T con el mismo esquema y con el conjunto de tuplas que pertenecen a R, a S o a ambas.



Notación: T = R U S

S = UNION(R, S)

C1	placa	marca	color
	`LAB384□	□ ford□	□ verde□
	□ LAM112□	□ toyota□	□ azul□
	'LGR889'	□ toyota□	'azul□

C2	placa	marca	color
	'LAB384`	□ ford□	□ verde□
	'XSG230`	□ ford□	□ gris□

C = C1 U C2

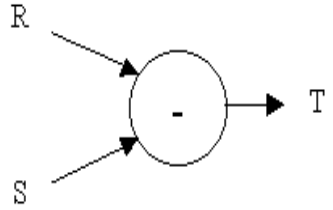
C	placa	marca	color
	`LAB384□	□ ford□	□ verde□
	□ LAM112□	□ toyota□	□ azul□
	□ LGR889□	□ toyota□	□ azul□
	'XSG230`	'ford'	'gris`

· **Diferencia:** (− or &setmn) La diferencia (R - S) de dos relaciones R y S con el mismo esquema es una relación T con el mismo esquema que contiene las tuplas que pertenecen a R y no pertenecen a S. (Es el conjunto de las tuplas que están en R pero no en S.).

Notación: T = R - S = MINUS(R, S)

C1	placa	marca	color
	`LAB384□	□ ford□	□ verde□
	□ LAM112□	□ toyota□	□ azul□
	'LGR889'	□ toyota□	´azul□

C2	placa	marca	color
	'LAB384'	□ ford□	□ verde□
	'XSG230'	□ ford□	□ gris□



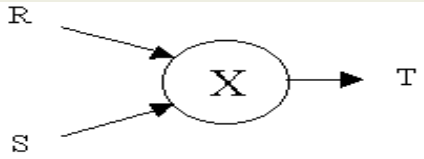
Ejemplo: C3 = C1 - C2

C3	placa	marca	color
	□ LAM112□	□ toyota□	□ azul□
	□ LGR889□	□ toyota□	□ azul□

- **Producto cartesiano:(x)** El producto cartesiano de dos relaciones R y S de cualquier esquema, es una relación T que contiene los atributos de R concatenados con los de S y sus tuplas son todas las formadas por la concatenación de una tupla de R con todas las tuplas de S.

Notación: T = R x S = PRODUCT(R, S)

C3	placa	marca	color
	□ LAM112□	□ toyota□	□ azul□
	□ LGR889□	□ toyota□	□ azul□



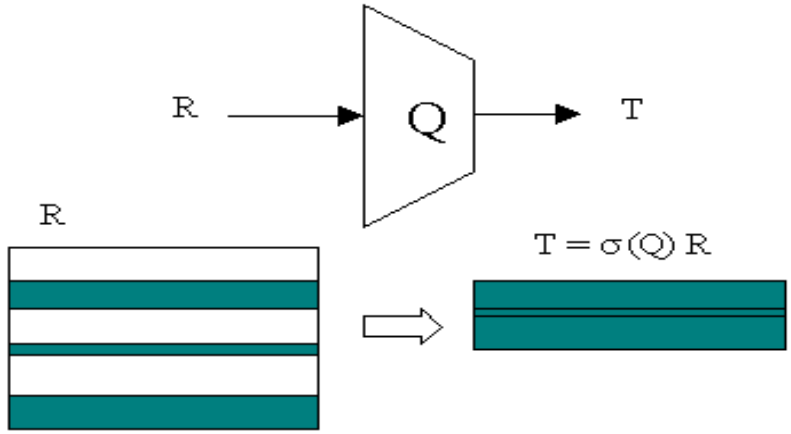
P1	ced	nom	ape
	□ 4411213□	□ Pedro□	□ Diaz□
	□ 8134977□	□ Juana□	□ Perez□

C4 = P1 x C3

C4	ced	nom	ape	placa	marca	color
	□ 4411213□	□ Pedro□	□ Diaz□	□ LAM112□	□ toyota□	□ azul□
	□ 4411213□	□ Pedro□	□ Diaz□	□ LGR889□	□ toyota□	□ azul□
	□ 8134977□	□ Juana□	□ Perez□	□ LAM112□	□ toyota□	□ azul□
	□ 8134977□	□ Juana□	□ Perez□	□ LGR889□	□ toyota□	□ azul□

- **Restricción:** La restricción de una relación R por un criterio de selección Q es una relación R y cuyas tuplas son aquellas que pertenecen a R y satisfacen Q. En Q los operandos pueden ser columnas o constantes y los operadores pueden ser de comparación, aritméticos y lógicos.

Notación: $s_Q (R) = RESTRICT (R / Q)$

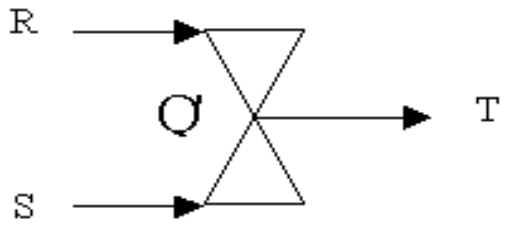


Ejemplo: R2 = C4 [ape < □ Mendez□]

R2	ced	nom	ape	placa	marca	color
	□ 4411213□	□ Pedro□	□ Diaz□	□ LAM112□	□ toyota□	□ azul□
	□ 4411213□	□ Pedro□	□ Diaz□	□ LGR889□	□ toyota□	□ azul□

- **Producto o conjunción (join) (∏) :** El producto de dos relaciones R y S según Q es un conjunto de tuplas del producto cartesiano R x S que satisfacen Q.
(También se le conoce como concatenación).

Notación: $T = R \ w_Q \ S = JOIN(R, S / Q)$



Numero Vendedor	Nombre Vendedor	Porcentaje comisión	Año contrato
137	Baker	10	1975
186	Adams	15	1971
204	Dickens	10	1963
361	Carlyle	20	1963

Número Cliente	Número Vendedor	Ciudad
121	137	Nueva York
839	186	Hartford
933	137	Boston
1047	137	Boston
1525	361	Newark
1700	361	Washington
1826	137	Nueva York
2198	204	Nueva York
2267	186	Nueva York

Numero Vendedor	Nombre Vendedo	Porcentaje comisión	Año contrato	Número Cliente	Numero Vendedor	Ciudad
137	Baker	10	1975	121	137	Nueva York
137	Baker	10	1975	993	137	Boston
137	Baker	10	1975	1047	137	Boston
137	Baker	10	1975	1826	137	Nueva York
186	Adams	15	1971	839	186	Hartford
186	Adams	15	1971	2267	186	Nueva York
204	Dickens	10	1963	2198	204	Nueva York
361	Carlyle	20	1963	1525	361	Newark
361	Carlyle	20	1963	1700	361	Washington

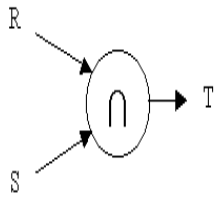
- **Intersección:** La intersección de dos relaciones R y S con el mismo esquema es una relación T con el mismo esquema que contiene las tuplas que pertenecen a R y a S a la vez.

Notación: T = R I S = INTERSECT(R, S)

T = R - (R - S) = S - (S - R)

C1	placa	marca	color
	`LAB384□	□ ford□	□ verde□
	□ LAM112□	□ toyota□	□ azul□
	'LGR889`	□ toyota□	´azul□

C2	placa	marca	color
	'LAB384`	□ ford□	□ verde□
	'XSG230`	□ ford□	□ gris□

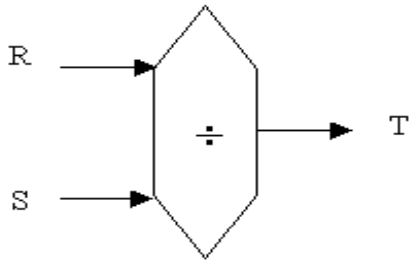


Ejemplo: T2 = C1 ⋂ C2

T2	placa	marca	color
	'LAB384□	□ ford□	□ verde□

- **División:** El cociente de $R(A_1, A_2, ..., A_n)$ por la subrelación $S(A_p, ..., A_n)$ es una relación $T(A_1, A_2, ..., A_{p-1})$ formada por las tuplas que concatenadas a cada una de las tuplas de S da **siempre** una tupla de R.

Notación: T = R ÷ S = DIVISION(R, S)



$R \div S = T - W$

$T = \Pi\ A_1, A_2, ..., A_{p-1}\ (R)$ $\quad W = \Pi\ A_1, A_2, ..., A_{p-1}\ ((T \times S) - R)$

Ejemplo:

C5	año	marca	potencia	color
	`1985□	□ ford□	1.8	□ verde□
	□ 1993□	□ toyota□	1.6	□ azul□
	□ 1993□	□ ford□	1.6	□ verde□
	‘1985`	□ toyota□	1.8	‘azul□

C6	año	potencia
	‘1985`	1.8
	‘1993`	1.6

T3 = C5 ÷ C6

T3	marca	color
	□ ford□	□ verde□
	□ toyota□	□ azul□

El álgebra relacional proporciona una serie de operaciones que se pueden usar para decir al sistema cómo *construir* la relación deseada a partir de las relaciones de la base de datos. El cálculo relacional proporciona una notación para formular la *definición* de la relación deseada en términos de las relaciones de la base de datos.

El cálculo relacional toma su nombre del *cálculo de predicados*, que es una rama de la lógica. Hay dos tipos de cálculo relacional, el *orientado a tuplas*, propuesto por Codd, y el *orientado a dominios*, propuesto por otros autores.

En el cálculo de predicados (lógica de primer orden), un *predicado* es una función con argumentos que se puede evaluar a verdadero o falso. Cuando los argumentos se sustituyen por valores, la función lleva a una expresión denominada *proposición*, que puede ser verdadera o falsa.

Por ejemplo, las frases `Carlos Baeza es un miembro de la plantilla' y `Carlos Baeza gana más que Amelia Pastor' son proposiciones, ya que se puede determinar si son verdaderas o falsas.

En el primer caso, la función `es un miembro de la plantilla' tiene un argumento (Carlos Baeza) y en el segundo caso, la función `gana más que' tiene dos argumentos (Carlos Baeza y Amelia Pastor).

Si un predicado tiene una variable, como en ` x es un miembro de la plantilla', esta variable debe tener un *rango* asociado. Cuando la variable se sustituye por alguno de los valores de su rango, la proposición puede ser cierta; para otros valores puede ser falsa.

Por ejemplo

Si el rango de x es el conjunto de todas las personas y reemplazamos x por Carlos Baeza, la proposición `Carlos Baeza es un miembro de la plantilla' es cierta. Pero si reemplazamos x por el nombre de una persona que no es miembro de la plantilla, la proposición es falsa.

Si F es un predicado, la siguiente expresión corresponde al conjunto de todos los valores de x para los que F es cierto:

x WHERE F(x)

Los predicados se pueden conectar mediante AND, OR y NOT para formar *predicados compuestos*.

Cálculo orientado a tuplas

En el cálculo relacional orientado a tuplas, lo que interesa es encontrar tuplas para las que se cumple cierto predicado. El cálculo orientado a tuplas se basa en el uso de *variables tupla*. Una variable tupla es una variable cuyo rango de valores son las tuplas de una relación.

Por ejemplo, para especificar el rango de la variable tupla PX sobre la relación PLANTILLA se utiliza la siguiente expresión:

RANGE OF PX IS PLANTILLA

Para expresar la consulta `obtener todas las tuplas PX para las que F(PX) es cierto', se escribe la siguiente expresión:

PX WHERE F(PX)

donde F es lo que se denomina una *fórmula bien formada (fbf)*. Por ejemplo, para expresar la consulta `obtener todos los datos de los empleados que ganan más de 10.000 pesos se puede escribir:

```
RANGE OF PX IS PLANTILLA
PX WHERE PX.salario > 10000
```

PX.salario se refiere al valor del atributo salario para la tupla PX. Para que se muestren solamente algunos atributos, por ejemplo, apellido y salario, en lugar de todos los atributos de la relación, se escribe:

```
RANGE OF PX IS PLANTILLA
PX.apellido, PX.salario WHERE PX.salario > 10000
```

Modificación de la Base de datos

Para la modificación de bases de datos se creo el SQL, esté cuenta con módulos DDL (Data definition Language), para la definición de datos que nos permite crear o modificar la estructura de las tablas.
Las instrucciones para realizar estas operaciones son:

CREATE TABLE: Nos permite crear una tabla de datos vacía.
INSERT: Permite almacenar registros en una tabla creada.
UPDATE: Permite modificar datos de registros almacenados en la tabla.
DELETE: Borra un registro entero o grupo de registros de una tabla.
CREATE INDEX: Crea un índice que nos puede auxiliar para las consultas.
DROP TABLE: Permite borrar una tabla.
DROP INDEX: Borra el índice indicado.

Estructura de la sentencia CREATE TABLE.

CREATE TABLE <Nombre de la tabla>
(
Atributo1: tipo de dato longitud **,**
Atributo2: tipo de dato longitud **,**
Atributo3: tipo de dato longitud **,**
:
:
Atributon: tipo de dato longitud **,**
PRIMARY KEY (Opcional)) **;**

Los campos pueden definirse como NOT NULL de manera opcional excepto en la llave primaria para lo cual es obligatorio. Además al definir la llave primaria se genera automáticamente un índice con respecto al campo llave; para definir la llave la denotamos dentro de los paréntesis de PRIMARY KEY.

Ejemplo:

Crear la tabla alumno, tomando como llave el numero de control.

```
CREATE TABLE Alumno
(
NControl char(8) NOT NULL,
NombreA char(20),
Especialidad char(3),
Dirección char(30),
PRIMARY KEY (NControl) );
```

Tabla Alumno:

NControl	NombreA	Especialidad	Dirección

Pueden existir más de una llave primaria, esto es si se requiere, se crearán tantos índices como llaves primarias se establezcan.

Pueden existir tantos campos Not Null (No nulos) como se requieran; En si estructurar la creación de una tabla es siempre parecida al ejemplo anterior.

* Estructura de la sentencia INSERT

INSERT

INTO Nombre de la tabla a la que se le va a insertar el registro

VALUES (Conjunto de valores del registro) **;**

Ejemplo:

Insertar en la tabla Alumno, antes creada los datos del alumno Daniel colín, con numero de control 95310518 de la especialidad de Ingeniería civil, con domicilio Abasolo Norte #45.

```
INSERT
INTO Alumno
VALUES("95310518","Daniel Colín","IC","Abasolo Norte #45") ;
```

Estructura de la Sentencia CREATE INDEX

CREATE INDEX Nombre que se le asignara al índice.

ON Nombre de la taba a la cual se le creara el índice (Campo(s) por el cual se creara el índice);

Ejemplo:

Crear un índice de la tabla Alumno por el campo Especialidad.

```
CREATE INDEX Indice1
ON Alumno(Especialidad);
```

Este índice contendrá a todos los alumnos ordenados por el campo especialidad.

```
CREATE INDEX UNIQUE INDEX Indice2
ON Alumno (Especialidad);
```

En la creación de este índice utilizamos la sentencia UNIQUE, es un indicador para permitir que se cree un índice único por especialidad, esta sentencia siempre se coloca antes de CREATE INDEX. En este ejemplo se creara un índice que contenga un alumno por especialidad existente.

* Estructura de la sentencia UPDATE

UPDATE Nombre de la tabla en donde se modificaran los datos.
SET Valores
WHERE (Condición);

Ejemplo:

Modificar el número de control del registro de Daniel Colín de la Tabla alumno por el número 96310518.

```
UPDATE Alumno
SET NControl  '96310518'
WHERE NombreA='Daniel Colín';
```

Estructura de la sentencia DROP TABLE

DROP TABLE Nombre de la tabla a borrar ;

Ejemplo:

Borrar la tabla Alumno creada anteriormente.

DROP TABLE Alumno;

Estructura de la sentencia DROP INDEX

DROP INDEX Nombre del índice a borrar;

Ejemplo:

Borrar el índice Indice1 creado anteriormente.

DROP INDEX Indice1;

* Estructura de la sentencia DELETE

DELETE
FROM Nombre de la tabla
WHERE Condición;

Ejemplos:

- Borrar el registro cuyo número de control es 95310386.

```
DELETE
FROM Alumno
WHERE Control='95310386';
```

- Borrar todos los registros de la tabla alumno.

```
DELETE
FROM Alumno;
```

En el primer ejemplo, se borrara todo el registro(todos los datos), del alumno con número de control = 95310386.

En el segundo ejemplo se borraran todos los registros de la tabla alumno, pero sin borrar la estructura de la tabla, ya que la orden Delete solo borra registros, la sentencia Drop Table es la que borra toda la estructura de la tabla junto con los registros de la misma.

Las ordenes que se utilizan para la manipulación de vistas son:
CREATE VIEW: Crea una tabla virtual.
DROP VIEW: Elimina una vista creada anteriormente.

CREATE VIEW Nombre de la vista **AS** (Expresión de consulta);

Para el ejemplos consideremos la tabla llamada CURSO, que contiene los siguientes campos: Estructura de la sentencia CREATE VIEW.

Nombre del campo	Descripción
NumC	Número del curso, único para identificar cada curso
NombreC	Nombre del curso, también es único
DescC	Descripción del curso
Creditos	Créditos, número de estos que gana al estudiante al cursarlo
Costo	Costo del curso.
Depto	Departamento académico que ofrece el curso.

Que contiene los siguientes datos:

Numc	NombreC	DescC	Creditos	Costo	Depto
A01	Liderazgo	Para público General	10	100.00	Admón.
S01	Introducción a la inteligencia artificial	Para ISC y LI	10	90.00	Sistemas.
C01	Construcción de torres	Para IC y Arquitectura	8	0.00	Ciencias
B01	Situación actual y perspectivas de la alimentación y la nutrición	Para IB	8	80.00	Bioquímica
E01	Historia presente y futuro de la energía solar	IE e II	10	100.00	Electromecánica.
S02	Tecnología OLAP	Para ISC y LI	8	100.00	Sistemas
C02	Tecnología del concreto y de las Estructuras	Para IC	10	100.00	Ciencias
B02	Metabolismo de lípidos en el camarón	Para IB	10	0.00	Bioquímica

Ejemplos:

* Crear una vista (tabla virtual), denominada CursosS, que contenga las filas solo correspondientes a cursos ofrecidos por el departamento Sistemas. La vista deberá contener todas las columnas de la tabla CURSO, con la excepción de la columna Depto, la secuencia, de izquierda a derecha de las columnas, deberá ser: NombreC, NumC, Creditos, Costo Y DescC.

CREATE VIEW CursosS **AS**
SELECT NombreC,NumC,Creditos,Costo,DescC
FROM CURSO
WHERE DescC='Sistemas';

Observemos que después del nombre de la vista ponemos la sentencia AS, esto para definir la estructura de la vista, la estructura en si de la vista esta formada por la consulta anteriormente vista utilizando la orden SELECT.

NombreC	NumC	Creditos	Costo	DescC
Introducción a la inteligencia artificial	S01	10	90.00	Para ISC y LI
Tecnología OLAP	S02	8	100.00	Para ISC y LI
Estructura de datos	S03	8	0.00	Para ISC y LI
Circuitos digitales	S04	10	0.00	Para ISC

Parte practica

Primeramente usaremos la sentencia SHOW para ver cuáles son las bases de datos existentes en el servidor al que estamos conectados:
mysql> SHOW DATABASES;

Si la base de datos "test" existe, hay que intentar accesar a ella:
mysql> USE test;

mysql> CREATE DATABASE zoologico;

mysql> USE zoológico;

Usaremos la sentencia CREATE TABLE para indicar como estarán conformados los registros de nuestras mascotas.
mysql> CREATE TABLE mascotas(
-> nombre VARCHAR(20), propietario VARCHAR(20),
-> especie VARCHAR(20), sexo CHAR(1), nacimiento DATE,
-> fallecimiento DATE);

Para verificar que la tabla fue creada como nosotros esperábamos, usaremos la sentencia DESCRIBE:

```
mysql> DESCRIBE mascotas;
```

```
Nombre Propietario Especie Sexo Nacimiento Fallecimiento
Fluffy Arnoldo Gato f 1999-02-04
Mau Juan Gato m 1998-03-17
Buffy Arnoldo Perro f 1999-05-13
FanFan Benito Perro m 2000-08-27
Kaiser Diana Perro m 1998-08-31 1997-07-29
Chispa Omar Ave f 1998-09-11
Wicho Tomás Ave m 2000-02-09
Skim Benito Serpiente m 2001-04-29
```

```
mysql> INSERT INTO mascotas
-> VALUES('Pelusa','Diana','Hamster','f','2000-03-30',NULL);
```

para agregar su registro en nuestra base de datos.

Recuperan todos los datos de una tabla:
mysql> SELECT * FROM mascotas;

Corregir sólo el registro erróneo con una sentencia UPDATE:
mysql> UPDATE mascotas SET nacimiento="1989-08-31" WHERE nombre="Kaiser";

seleccionamos sólo el registro de Kaiser de la siguiente manera:
mysql> SELECT * FROM mascotas WHERE nombre="Kaiser";

mysql> SELECT * FROM mascotas WHERE nacimiento >= "2000-1-1";

mysql> SELECT * FROM mascotas WHERE especie="Perro" AND sexo="f";

mysql> SELECT * FROM mascotas WHERE;

mysql> SELECT * FROM mascotas WHERE (especie = "Gato" AND sexo = "m") or (especie = "Ave" OR especie = "Gato");

```
mysql> SELECT nombre, nacimiento FROM mascotas;

mysql> SELECT propietario FROM mascotas;

mysql> SELECT DISTINCT propietario FROM mascotas;

mysql> SELECT nombre, especie, nacimiento FROM mascotas
-> WHERE especie = "perro" OR especie = "gato";

mysql> SELECT nombre, nacimiento FROM mascotas ORDER BY nacimiento;

mysql> SELECT nombre, nacimiento FROM mascotas ORDER BY
-> nacimiento DESC;

mysql> SELECT nombre, especie, nacimiento FROM mascotas
-> ORDER BY especie, nacimiento DESC;
```

Para determinar la edad de cada una de nuestras mascotas,

```
mysql> SELECT nombre, nacimiento, CURRENT_DATE,
-> (YEAR(CURRENT_DATE) - YEAR(nacimiento))
-> - (RIGHT(CURRENT_DATE,5) < RIGHT(nacimiento,5)) AS edad FROM mascotas;
```

Por ejemplo, para ordenar por nombre, usaremos la siguiente consulta:

```
mysql> SELECT nombre, nacimiento, CURRENT_DATE,
-> (YEAR(CURRENT_DATE) - YEAR(nacimiento))
-> - (RIGHT(CURRENT_DATE,5) < RIGHT(nacimiento,5))
-> AS edad FROM mascotas ORDER BY nombre;
```

```
mysql> SELECT nombre, nacimiento, CURRENT_DATE,
-> (YEAR(CURRENT_DATE) - YEAR(nacimiento))
-> - (RIGHT(CURRENT_DATE,5) < RIGHT(nacimiento,5))
-> AS edad FROM mascotas ORDER BY edad;
```

Para encontrar los nombres que comienzan con b:
mysql> SELECT * FROM mascotas WHERE nombre LIKE "b%";

Para encontrar los nombres que finalizan con fy:
mysql> SELECT * FROM mascotas WHERE nombre LIKE "%fy";

Para encontrar nombres que contienen una s:
mysql> SELECT * FROM mascotas WHERE nombre LIKE "%s%";

Contar el número total de animalitos
mysql> SELECT COUNT(*) FROM mascotas;

Si deseamos conocer cuántas mascotas tiene cada uno de los propietarios, la consulta es la siguiente:

```
mysql> SELECT propietario, COUNT(*) FROM mascotas GROUP BY propietario
```

El número de animalitos por especie:

```
mysql> SELECT especie, COUNT(*) FROM mascotas GROUP BY especie ;
```

El número de animalitos por sexo:

```
mysql> SELECT sexo, COUNT(*) FROM mascotas GROUP BY sexo:
```

El número de animalitos por combinación de especie y sexo:

```
mysql> SELECT especie, sexo, COUNT(*) FROM mascotas GROUP BY especie, sexo
```

para ver únicamente los datos de perritos y gatitos

```
mysql> SELECT especie, sexo, COUNT(*) FROM mascotas
-> WHERE especie="Perro" OR especie="Gato"
-> GROUP BY especie, sexo;
```

SQL con otra base de datos

Los valores que podemos guardar son:
A) TEXTO: Para almacenar texto usamos cadenas de caracteres. Las cadenas se colocan entre comillas simples. Podemos almacenar dígitos con los que no se realizan operaciones matemáticas, por ejemplo, códigos de identificación, números de documentos, números telefónicos. Tenemos los siguientes tipos: varchar, char y text.
B) NUMEROS: Existe variedad de tipos numéricos para representar enteros, negativos, decimales. Para almacenar valores enteros, por ejemplo, en campos que hacen referencia a cantidades, precios, etc., usamos el tipo integer. Para almacenar valores con decimales utilizamos: float o decimal.
C) FECHAS Y HORAS: para guardar fechas y horas dispone de varios tipos: date (fecha), datetime (fecha y hora), time (hora), year (año) y timestamp.
D) OTROS TIPOS: enum y set representan una enumeración y un conjunto respectivamente. Lo veremos más adelante.
E) Otro valor que podemos almacenar es el valor "null". El valor 'null' significa “valor desconocido” o "dato inexistente", ya lo estudiamos. No es lo mismo que 0 o una cadena vacía.

UAEM | Universidad Autónoma del Estado de México

```
drop table if exists libros;

create table libros(
  codigo integer unsigned auto_increment,
  titulo varchar(20) not null,
  autor varchar(30),
  editorial varchar(15),
  precio float unsigned,
  cantidad integer unsigned,
  primary key (codigo)
 );
```

```
create table libros(
codigo int unsigned auto_increment,
titulo varchar(40) not null,
editorial varchar(15),
autor varchar(30) default 'Desconocido',
precio decimal(5,2) unsigned default 1.11,
cantidad mediumint unsigned not null, primary key (codigo) );
```

Ingresamos algunos registros omitiendo algunos campos para ver cómo trabaja la cláusula "default".

```
insert into libros (titulo,autor,editorial,precio,cantidad) values('Java en 10 minutos','Juan Pereyra','Paidos',25.7,100);
```

SELECT "nombre_de_atributo" FROM "nombre_tabla";

Sueldos

Nombre	Sueldo	Fecha
Los Angeles	1500	05-Jan-1999
San Diego	250	07-Jan-1999
Los Angeles	300	08-Jan-1999
Boston	700	08-Jan-1999

SELECT Nombre FROM Sueldos;

Nombre
Los Angeles
San Diego
Los Angeles
Boston

Memory table

SELECT ALL FROM Sueldos;

SELECT * FROM Sueldos;

SELECT Nombre FROM Salarios WHERE Sueldo > 1000;

Nombre
Los Angeles

SELECT Nombre FROM Salarios WHERE Sueldo > 1000 OR (Sueldo < 500 AND Sueldo > 275);

Nombre
Los Angeles
San Francisco

SELECT * FROM Salarios WHERE Nombre IN ('Los Angeles', 'San Diego');

Nombre	Salario	Fecha
Los Angeles	1500	05-Jan-1999
San Diego	250	07-Jan-1999

<expresión1>	Operador	<expresión2>	Resultado
Verdad	AND	Falso	Falso
Verdad	AND	Verdad	Verdad
Falso	AND	Verdad	Falso
Falso	AND	Falso	Falso
Verdad	OR	Falso	Verdad
Verdad	OR	Verdad	Verdad
Falso	OR	Verdad	Verdad
Falso	OR	Falso	Falso
Verdad	XOR	Verdad	Falso
Verdad	XOR	Falso	Verdad
Falso	XOR	Verdad	Verdad
Falso	XOR	Falso	Falso
Verdad	Eqv	Verdad	Verdad
Verdad	Eqv	Falso	Falso
Falso	Eqv	Verdad	Falso
Falso	Eqv	Falso	Verdad
Verdad	Imp	Verdad	Verdad
Verdad	Imp	Falso	Falso
Verdad	Imp	Null	Null
Falso	Imp	Verdad	Verdad
Falso	Imp	Falso	Verdad
Falso	Imp	Null	Verdad
Null	Imp	Verdad	Verdad
Null	Imp	Falso	Null
Null	Imp	Null	Null

SELECT DISTINCT nombre FROM Salarios;

Nombre
Los Angeles
San Diego
Boston

SELECT * FROM Salarios WHERE fecha BETWEEN '06-Jan-1999' AND '10-Jan-1999';

Nombre	Salario	Fecha
San Diego	250	07-Jan-1999
San Francisco	300	08-Jan-1999
Boston	700	08-Jan-1999

SELECT * FROM Salarios WHERE Nombre LIKE '%AN%';

Nombre	Salario	Fecha
LOS ANGELES	1500	05-Jan-1999
SAN DIEGO	250	07-Jan-1999
SAN FRANCISCO	300	08-Jan-1999

SELECT Nombre, Salario, fecha FROM Salarios ORDER BY Salario DESC;

Nombre	Salario	Fecha
Los Angeles	1500	05-Jan-1999
Boston	700	08-Jan-1999
San Francisco	300	08-Jan-1999
San Diego	250	07-Jan-1999

Funciones

AVG
COUNT
MAX
MIN
SUM

La sintaxis para el uso de funciones es,

SELECT "tipo de función"("nombre_atributo") FROM "nombre_tabla";

SELECT SUM(Salario) FROM Sueldos;

SUM(Salario)
2750

SELECT Nombre, Telefono FROM Clientes;

SELECT CodigoPostal, Nombre, Telefono FROM Clientes ORDER BY Nombre;

Consulta devuelve los atributosos CodigoPostal, Nombre, Telefono de la tabla Clientes ordenados por el campo Nombre.

SELECT CodigoPostal, Nombre, Telefono FROM Clientes ORDER BY Nombre;

Se pueden ordenar los registros por mas de un campo, como por ejemplo:

SELECT CodigoPostal, Nombre, Telefono FROM Clientes ORDER BY CodigoPostal, Nombre;

Incluso se puede especificar el orden de los registros: ascendente mediante la claúsula (ASC -se toma este valor por defecto) ó descendente (DESC)

SELECT CodigoPostal, Nombre, Telefono FROM Clientes ORDER BY CodigoPostal DESC , Nombre ASC;

```
SELECT * FROM Empleados WHERE Edad > 25 AND Edad < 50;

SELECT * FROM Empleados WHERE (Edad > 25 AND Edad < 50) OR Sueldo = 100;

SELECT * FROM Empleados WHERE NOT Estado = 'Soltero';

SELECT * FROM Empleados WHERE (Sueldo > 100 AND Sueldo < 500) OR  Provincia = 'Madrid' AND Estado = 'Casado');

SELECT * FROM Pedidos WHERE CodPostal Between 28000 And 28999;

SELECT * FROM Pedidos WHERE Provincia In ('Madrid', 'Barcelona', 'Sevilla');

SELECT * FROM Pedidos WHERE Fecha_Envio = #5/10/94#;

SELECT Apellidos, Nombre FROM Empleados WHERE Apellidos = 'King';

SELECT Apellidos, Nombre FROM Empleados WHERE Apellidos Like 'S*';

SELECT Apellidos, Salario FROM Empleados WHERE Salario Between 200 And 300;
```

SELECT Apellidos, Salario FROM Empl WHERE Apellidos Between 'Lon' And 'Tol';

SELECT Id_Pedido, Fecha_Pedido FROM Pedidos WHERE Fecha_Pedido Between #1-1-94# And #30-6-94#;

SELECT Apellidos, Nombre, Ciudad FROM Empleados WHERE Ciudad In ('Sevilla', 'Los Angeles', 'Barcelona');

ALIAS

nombre	apellido1	apellido2
ANTONIO	PEREZ	GOMEZ
ANTONIO	GARCIA	RODRIGUEZ
PEDRO	RUIZ	GONZALEZ

SELECT personas.apellido1, personas.apellido2 FROM personas WHERE personas.nombre = 'ANTONIO';

apellido1	apellido2
PEREZ	GOMEZ
GARCIA	RODRIGUEZ

Utilizamos el alias 'p' para la tabla 'personas', para simplificar la sentencia SELECT

SELECT p.apellido1, p.apellido2 FROM personas AS p WHERE p.nombre = 'ANTONIO';

La sentencia **INNER JOIN** es el sentencia JOIN por defecto, y consiste en combinar cada fila de una tabla con cada fila de la otra tabla, seleccionado aquellas filas que cumplan una determinada condición.

per	nombre	apellido1	apellido2	dep
1	ANTONIO	PEREZ	GOMEZ	1
2	ANTONIO	GARCIA	RODRIGUEZ	2
3	PEDRO	RUIZ	GONZALEZ	2

dep	departamento
1	ADMINISTRACION
2	INFORMATICA
3	COMERCIAL

SELECT nombre, apellido1, departamento FROM personas INNER JOIN departamentos WHERE personas.dep = departamentos.dep

nombre	apellido1	departamento
ANTONIO	PEREZ	ADMINISTRACION
ANTONIO	GARCIA	INFORMATICA
PEDRO	RUIZ	INFORMATICA

La sentencia **LEFT JOIN** combina los valores de la primera tabla con los valores de la segunda tabla. Siempre devolverá las filas de la primera tabla, incluso aunque no cumplan la condición.

per	nombre	apellido1	apellido2	dep
1	ANTONIO	PEREZ	GOMEZ	1
2	ANTONIO	GARCIA	RODRIGUEZ	2
3	PEDRO	RUIZ	GONZALEZ	2

dep	departamento
1	ADMINISTRACION
2	INFORMATICA
3	COMERCIAL

```
SELECT nombre, apellido1, departamento
FROM personas
LEFT JOIN departamentos
WHERE personas.dep = departamentos.dep;
```

nombre	apellido1	departamento
ANTONIO	PEREZ	ADMINISTRACION
ANTONIO	GARCIA	INFORMATICA
PEDRO	RUIZ	

Aunque el departamento '4' de PEDRO RUIZ no existe en la tabla de departamentos, devolverá la fila con esa columna 'departamento' en blanco.

La sentencia **RIGHT JOIN** combina los valores de la primera tabla con los valores de la segunda tabla. Siempre devolverá las filas de la segunda tabla, incluso aunque no cumplan la condición.

En algunas bases de datos, la sentencia RIGHT JOIN es igual a RIGHT OUTER JOIN

EJEMPLO SQL RIGHT JOIN

per	nombre	apellido1	apellido2	dep
1	ANTONIO	PEREZ	GOMEZ	1
2	ANTONIO	GARCIA	RODRIGUEZ	2
3	PEDRO	RUIZ	GONZALEZ	4

dep	departamento
1	ADMINISTRACION
2	INFORMATICA
3	COMERCIAL

```
SELECT nombre, apellido1, departamento
FROM personas
RIGHT JOIN departamentos
WHERE personas.dep = departamentos.dep;
```

nombre	apellido1	departamento
ANTONIO	PEREZ	ADMINISTRACION
ANTONIO	GARCIA	INFORMATICA
		COMERCIAL

Aunque no exista ninguna persona del departamento 'COMERCIAL' (3), esta fila aparecerá con las otras columnas en blanco

La sentencia **FULL JOIN** combina los valores de la primera tabla con los valores de la segunda tabla. Siempre devolverá las filas de las dos tablas, aunque no cumplan la condición.
La sentencia FULL JOIN es la unión de LEFT JOIN y RIGHT JOIN

per	nombre	apellido1	apellido2	dep
1	ANTONIO	PEREZ	GOMEZ	1
2	ANTONIO	GARCIA	RODRIGUEZ	2
3	PEDRO	RUIZ	GONZALEZ	4

dep	departamento
1	ADMINISTRACION
2	INFORMATICA
3	COMERCIAL

```
SELECT nombre, apellido1, departamento
FROM personas
FULL JOIN departamentos
WHERE personas.dep = departamentos.dep;
```

nombre	apellido1	departamento
ANTONIO	PEREZ	ADMINISTRACION
ANTONIO	GARCIA	INFORMATICA
PEDRO	RUIZ	COMERCIAL

La sentencia SQL **UNION** es utilizada para acumular los resultados de dos sentencias SELECT. Las dos sentencias SELECT tienen que tener el mismo número de columnas, con el mismo tipo de dato y en el mismo orden.

Tabla "personas_empresa1"

per	nombre	apellido1	apellido2
1	ANTONIO	PEREZ	GOMEZ
2	ANTONIO	GARCIA	RODRIGUEZ
3	PEDRO	RUIZ	GONZALEZ

Tabla "personas_empresa2"

per	nombre	apellido1	apellido2
1	JUAN	APARICIO	TENS
2	ANTONIO	GARCIA	RODRIGUEZ
3	LUIS	LOPEZ	VAZQUEZ

```
SELECT nombre, apellido1 FROM personas_empresa1
UNION
SELECT nombre, apellido1 FROM personas_empresa2
```

nombre	apellido1
ANTONIO	PEREZ
ANTONIO	GARCIA
PEDRO	RUIZ
JUAN	APARICIO
LUIS	LOPEZ

La persona 'ANTONIO GARCIA RODRIGUEZ' aparecerá solo una vez en el resultado, porque no aparecerán las filas repetidas.

La clave externa o **FOREIGN KEY**, es una columna o varias columnas, que sirven para señalar cual es la clave primaria de otra tabla.
La columna o columnas señaladas como **FOREIGN KEY**, solo podrán tener valores que ya existan en la clave primaria PRIMARY KEY de la otra tabla.
Ejemplo de **FOREIGN KEY**
Tabla "departamentos", con la clave primaria "dep"

dep	departamento
1	ADMINISTRACION
2	INFORMATICA
3	COMERCIAL

Tabla personas, con una clave externa **FOREIGN KEY** 'dep', que hace referencia a la clave primaria 'dep' de la tabla anterior 'departamentos' y por tanto, solo puede tener un valor de los que tiene en esa tabla

per	nombre	apellido1	apellido2	dep
1	ANTONIO	PEREZ	GOMEZ	1
2	ANTONIO	GARCIA	RODRIGUEZ	2
3	PEDRO	RUIZ	GONZALEZ	4

CREATE TABLE departamentos { dep int NOT NULL, departamento varchar(255), PRIMARY KEY (dep) }

CREATE TABLE personas { per int NOT NULL, nombre varchar(255), apellido1 varchar(255), dep int NOT NULL, PRIMARY KEY (per), FOREIGN KEY (dep) REFERENCES departamentos(dep) }

Lecturas y practica recomendada

http://www.emagister.com/curso-mysql-php/caracteristicas-mas-significativas-sql

http://www.lawebdelprogramador.com/cursos/archivos/ManualPracticoSQL.pdf

http://www.youtube.com/watch?v=HO5eb2wBaBk

http://www.youtube.com/watch?v=6n1lmCyfqLU

http://www.monografias.com/trabajos11/manu/manu.shtml

http://www.1keydata.com/es/sql/sql-fonciones.php

https://mariadb.com/kb/es/comandos-sql-basicos/

http://sql.11sql.com/